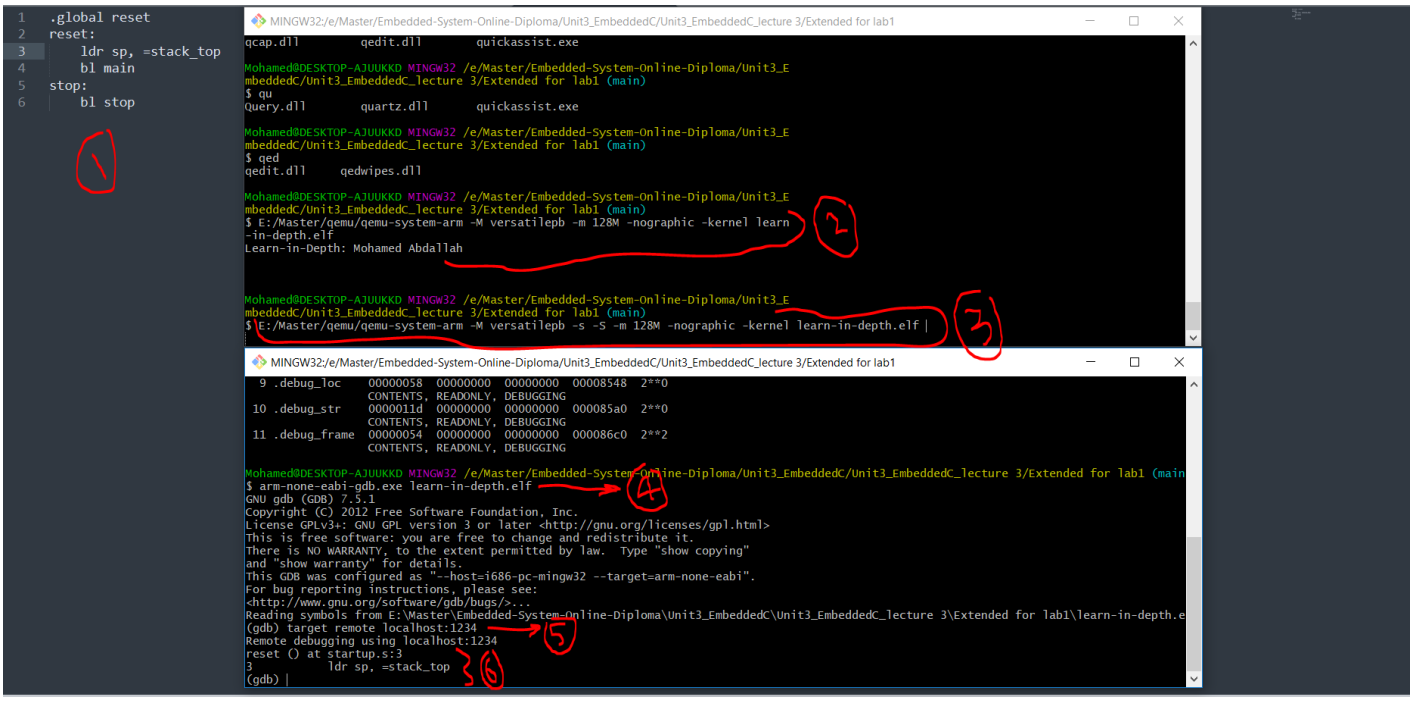
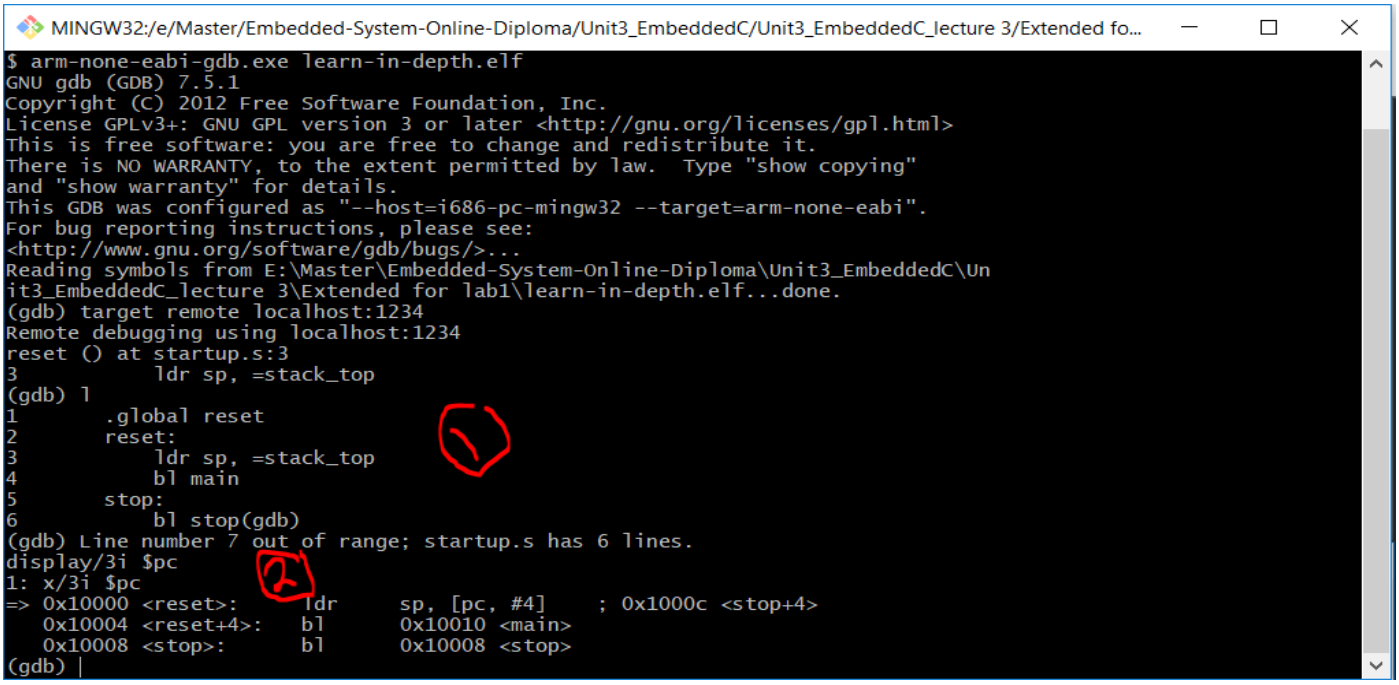
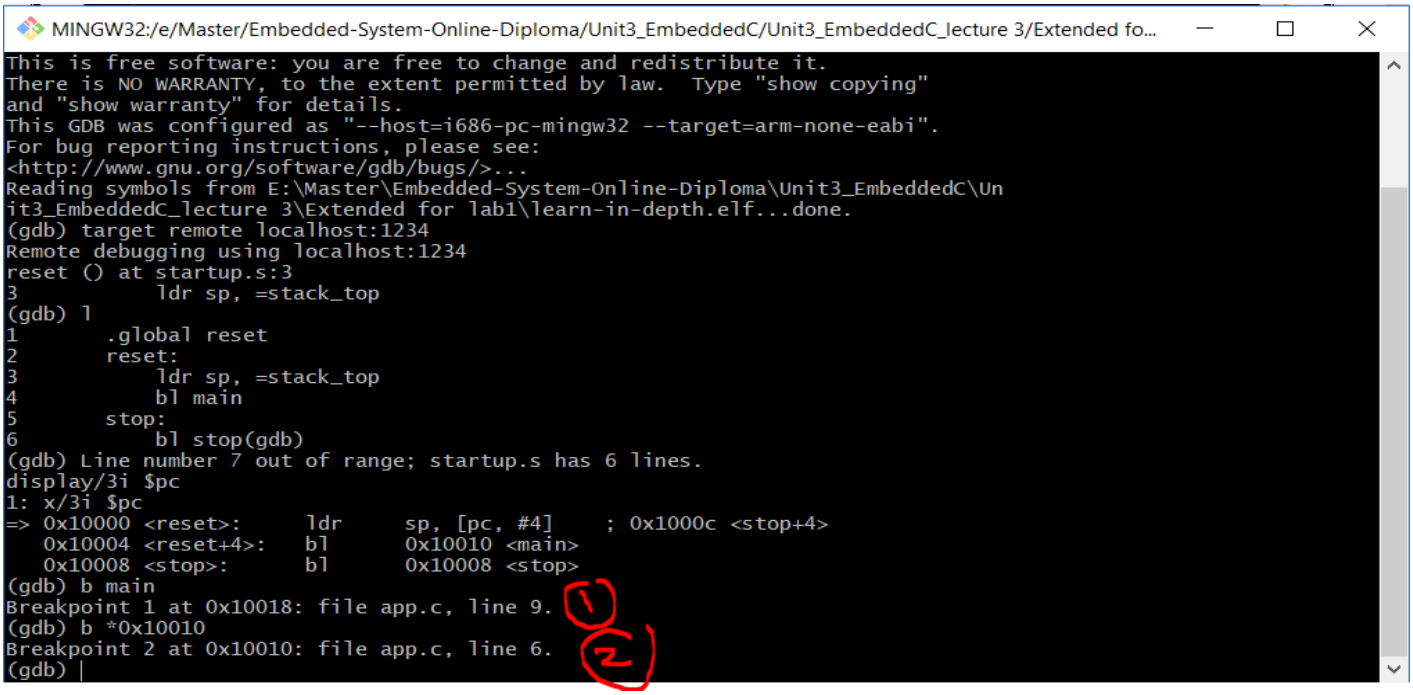
REPORT
FOR
EXTENDED
LAB1



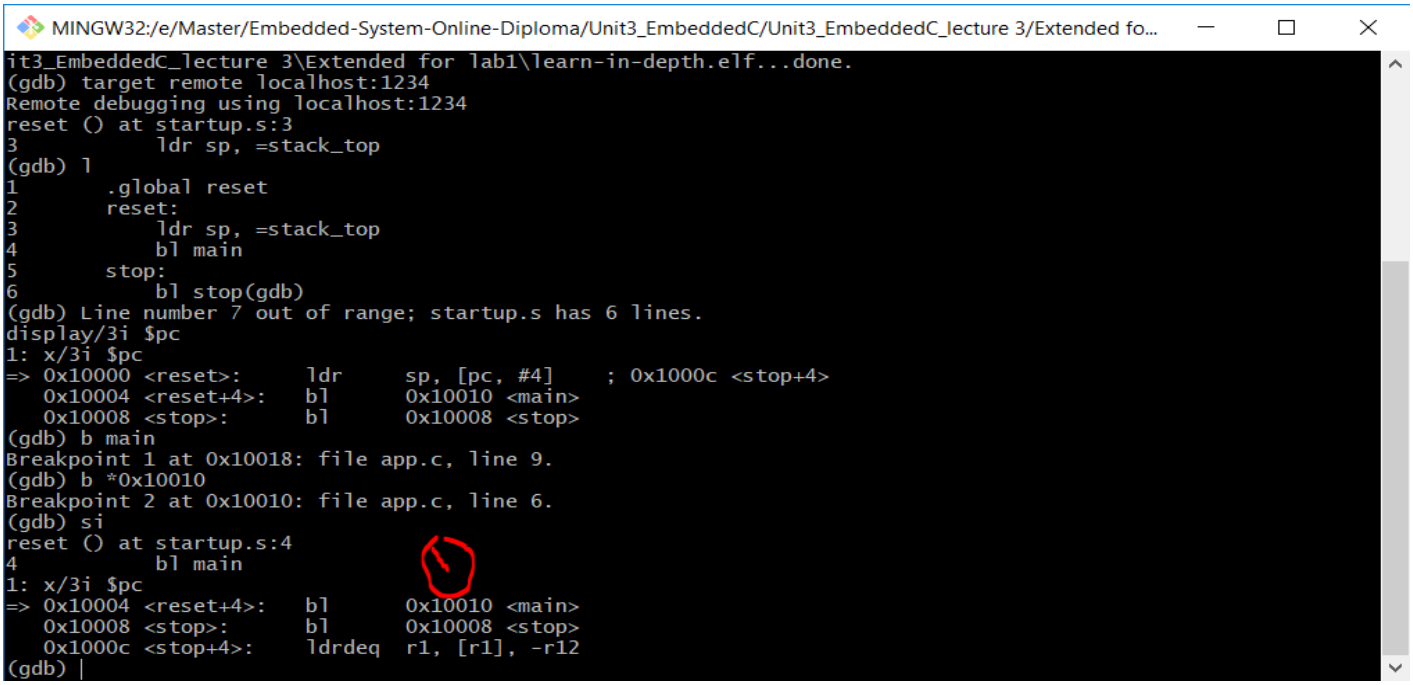
1. Startup.s file (assembly).
2. Turning on our board (versatilepb) and burning our barmetal software application (writing on uart Learn-in-Depth: Mohamed Abdallah).
3. My board is ready and wait debugger.
4. Path learn-in-depth.elf to debug (not .bin as it's not contain debug info).
5. Open path with my board (localhost → my machine IP, 1234 → port, info from board datasheet).
6. Processor @ entry point (reset section at startup file loading value of stack_top to stack pointer "initializing stack").



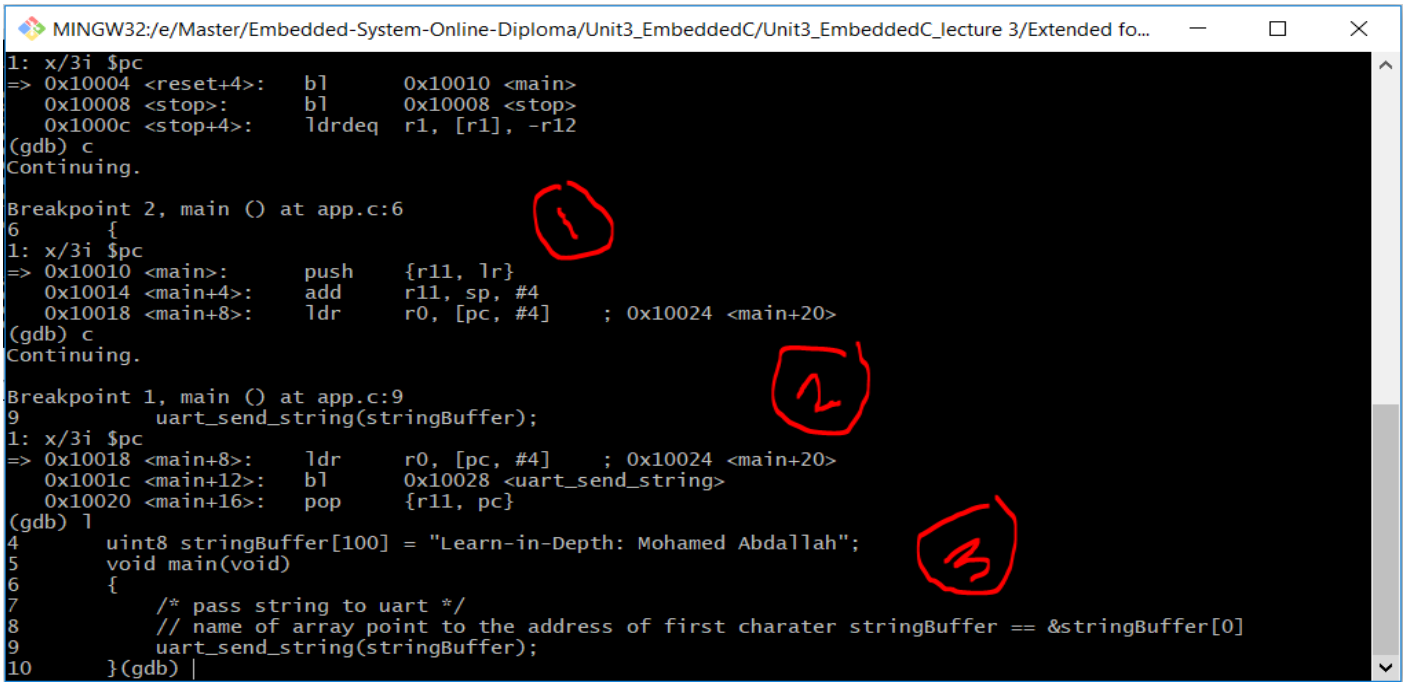
1. (l) show currently file.
2. (display/3i $pc) show three instructions, currently program counter point to next instruction should be fetched (0x10000 initializing stack then fetch next instruction which will branch to main).



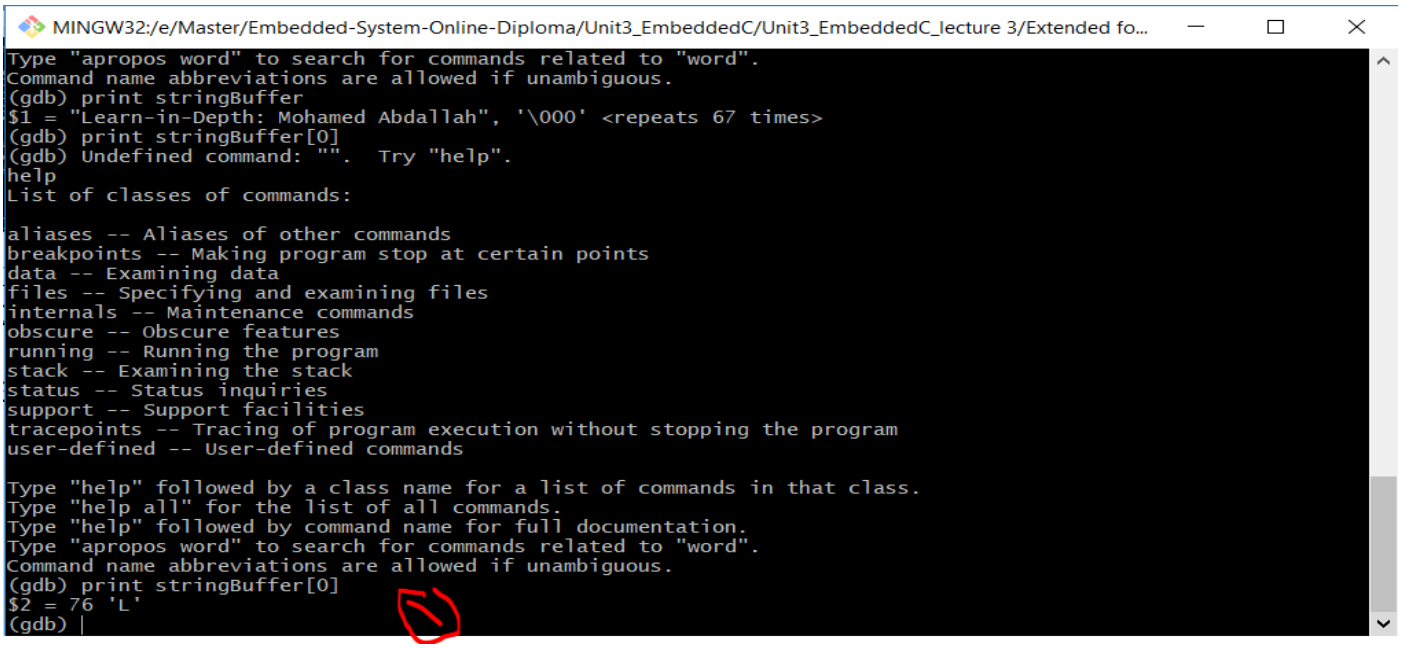
1. (b main) breakpoint at main function (0x10018 first line of c code in main between 0x10010 & 0x10018 context switching and initializing stack of main).
2. (b *0x10010) breakpoint at address 0x10010 to watch context and initializing of main stack.



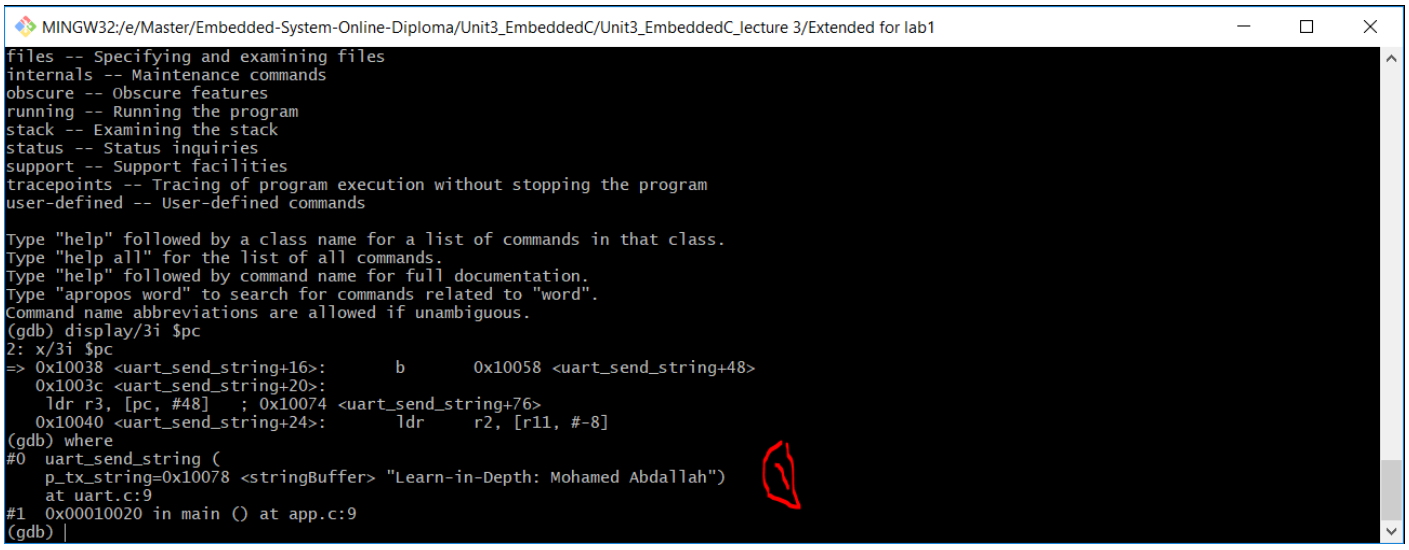
1. (si → step instruction) pc will point to next address 0x10004 instruction which will be fetched.
   (s → step c code level one line code in c maybe equivalent to many assembly instruction so).



1. (c → continue) will jump to first breakpoint at address 0x10010.
2. (c) again will jump to next breakpoint at address 0x10018 which first c code of main function at line 9.
3. (l show currently file app.c) we found that first line of c main at line 9 already.

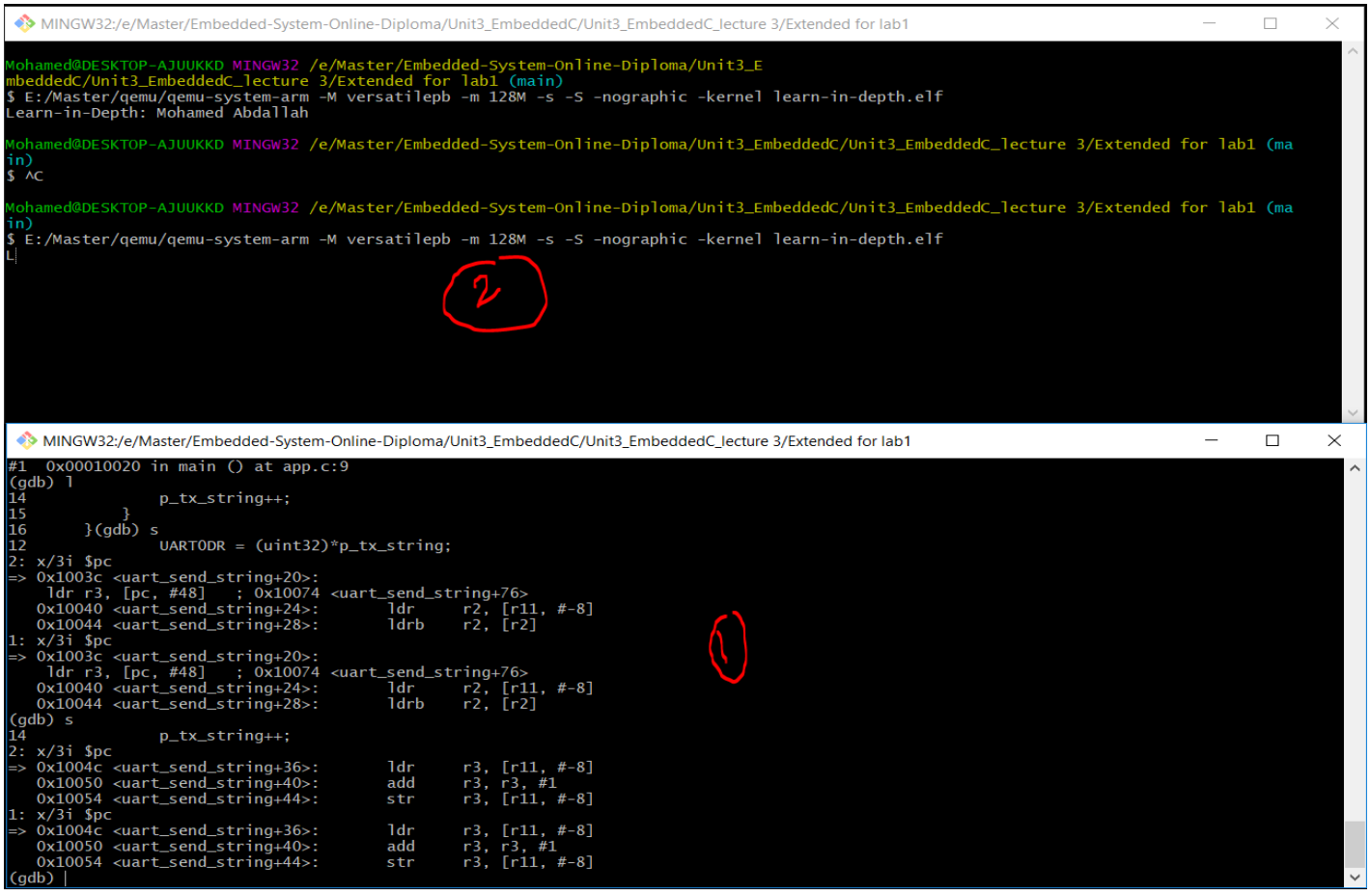


1. (print stringBuffer[0]) print value of variable 76 asci code.



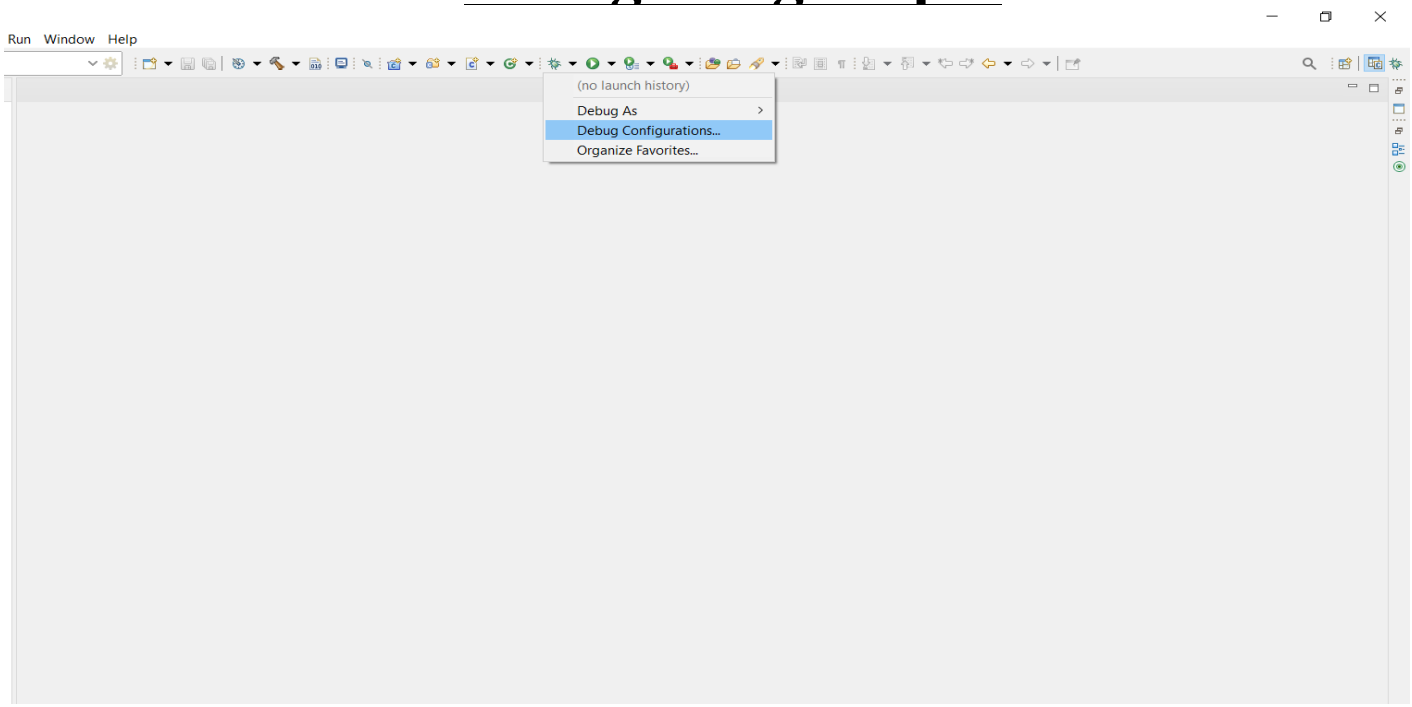
- I set breakpoint at uart_send_string then pressed c to jump to it.

1. (where) currently at uart_send_string function in uart.c file and reached here by call was in main function in file app.c at line 9.

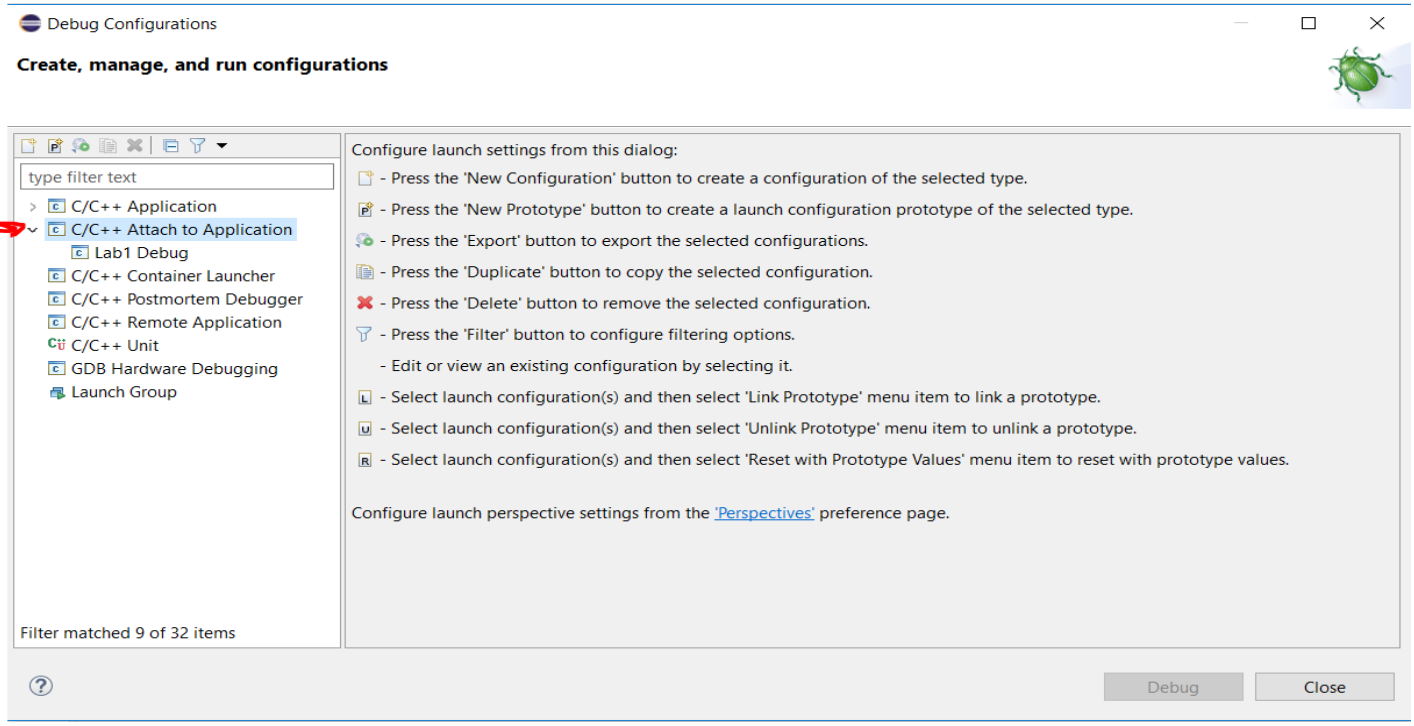


1. Using s will move line by line in c code to watch uart displaying our string written character by character.

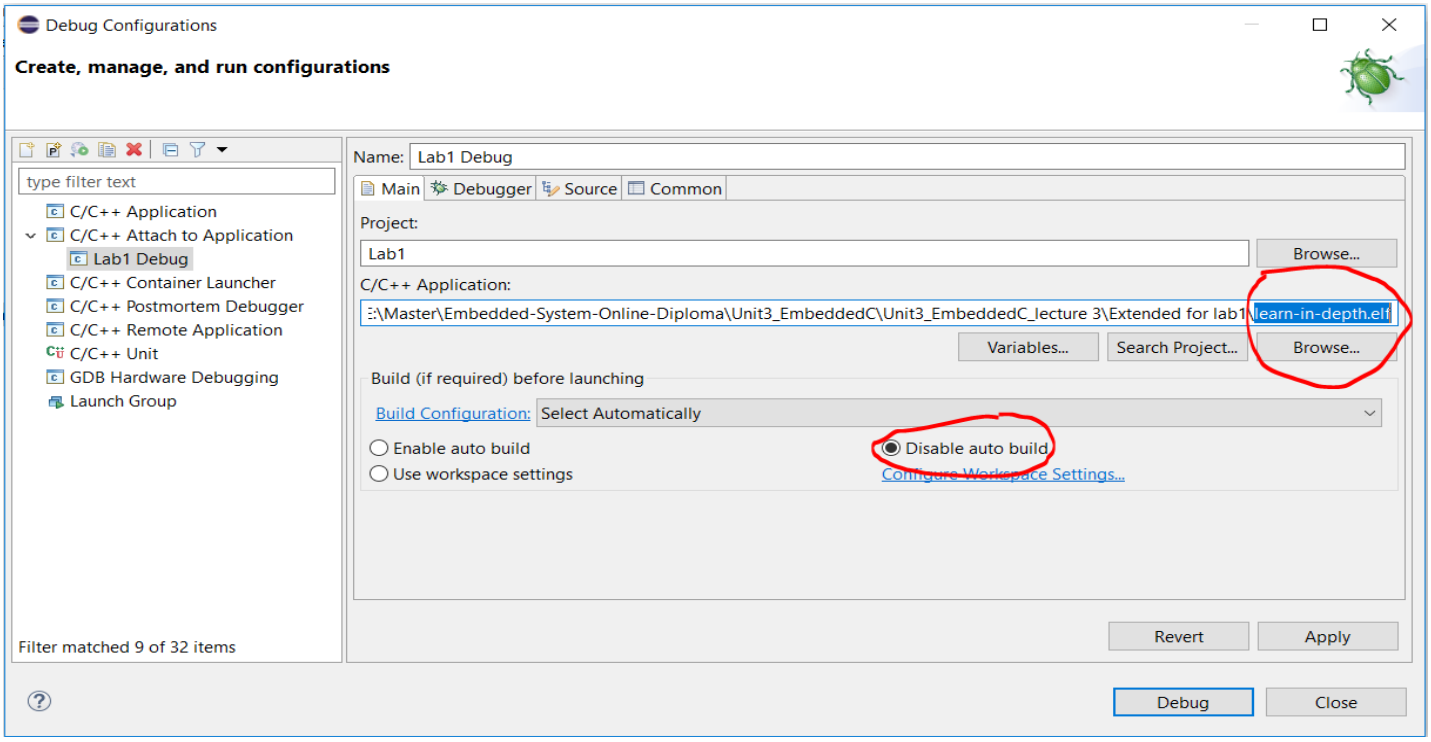
Debug using eclipse



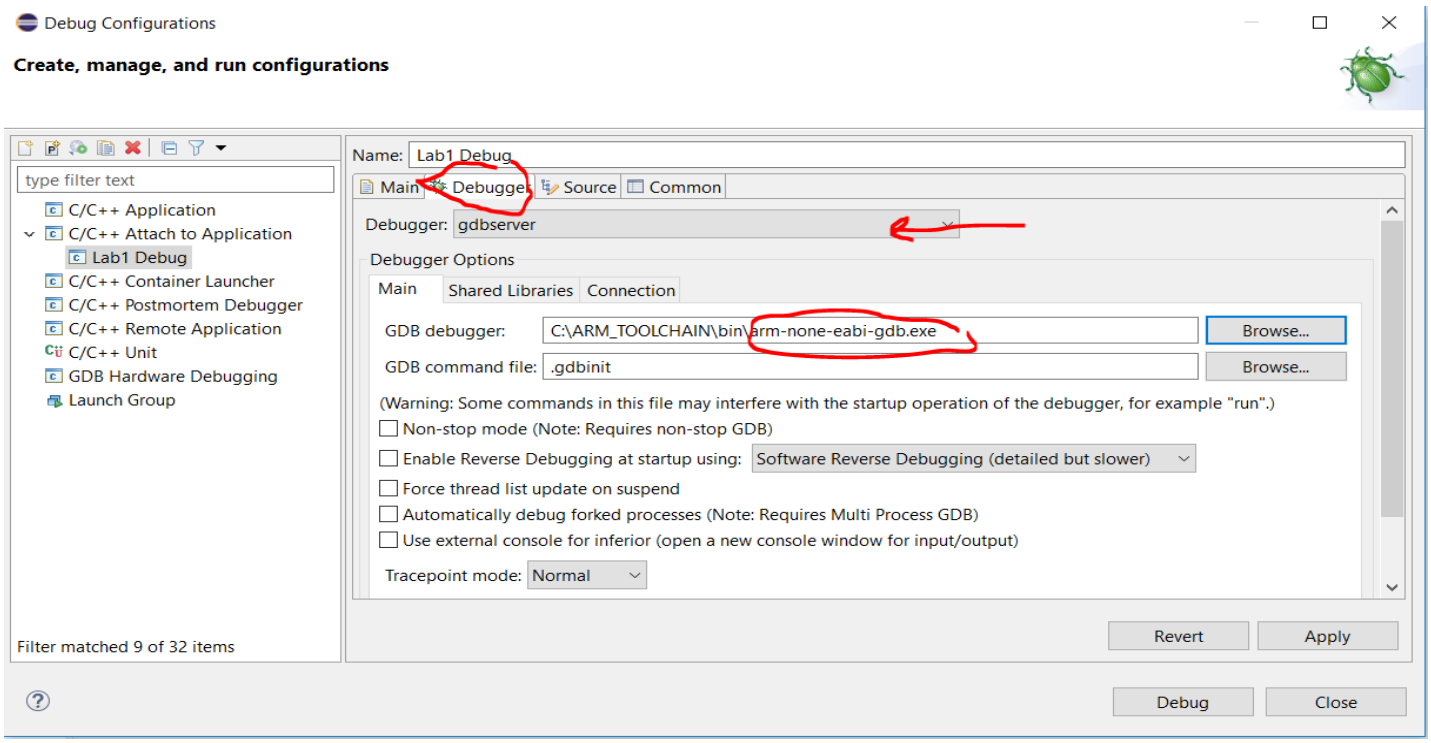
- Open empty project then select debug config.



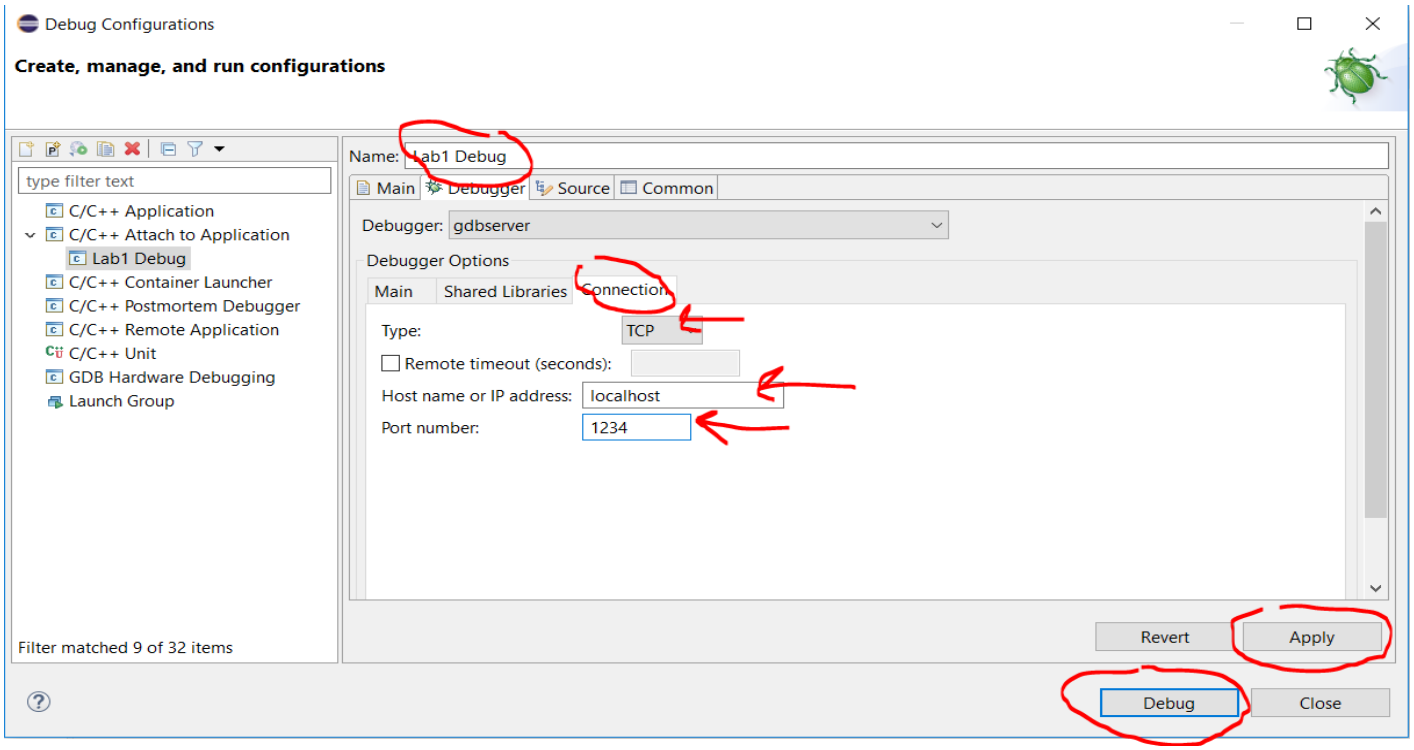
- Then dapple click on attach to application.



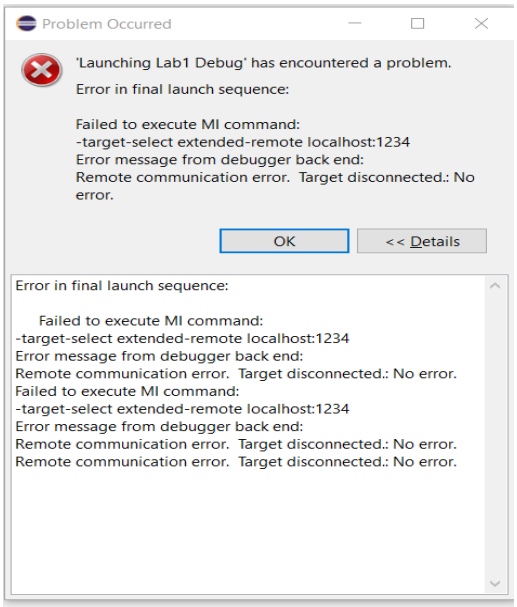
- Then select my .elf file and disable auto bulid.



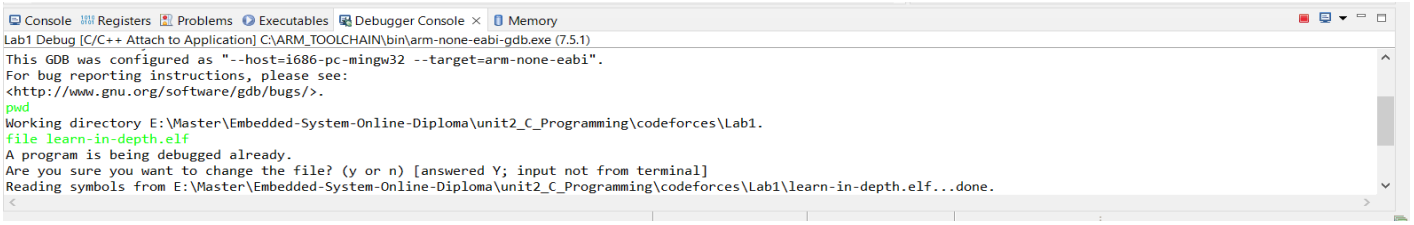
- Then select debugger tab, select gdbserver then select gdb cross tool chain not native as our code will not run on our pc it will run on anther machine.



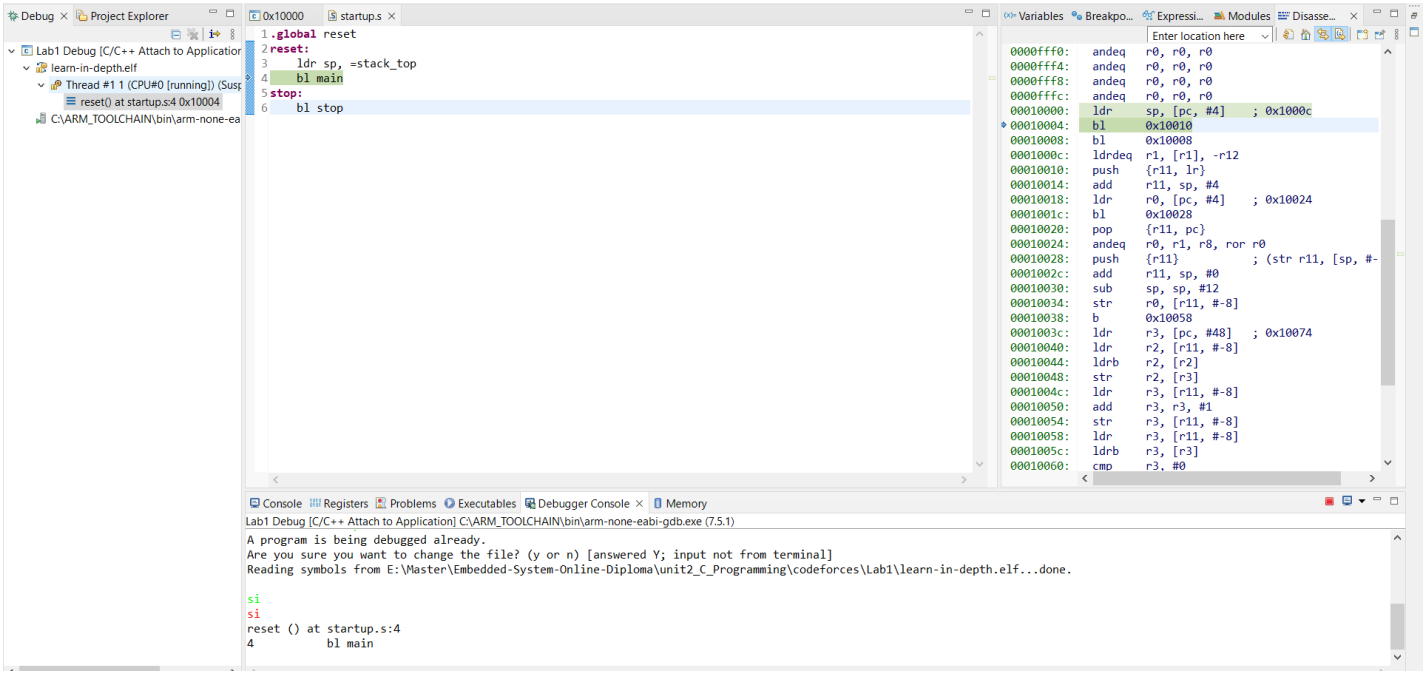
- Select connection tab and select Type and set IP and port number.
- Give name to that debug config then applay and Debug.



- Maybe face this problem if you forgot to Open your board.

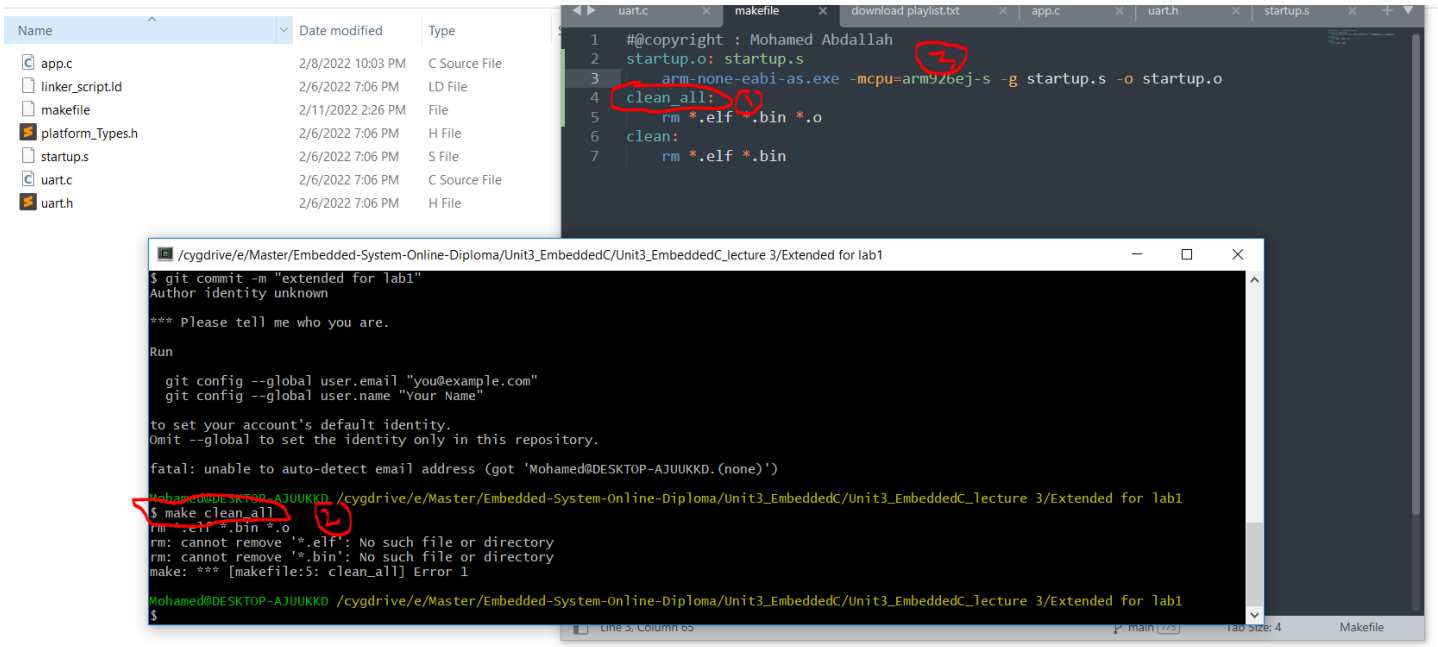


- You need to know your current directory by using pwd and copy your .elf there then write this command file file-name.elf.
- You will see that gdb can now read symbols successfully.

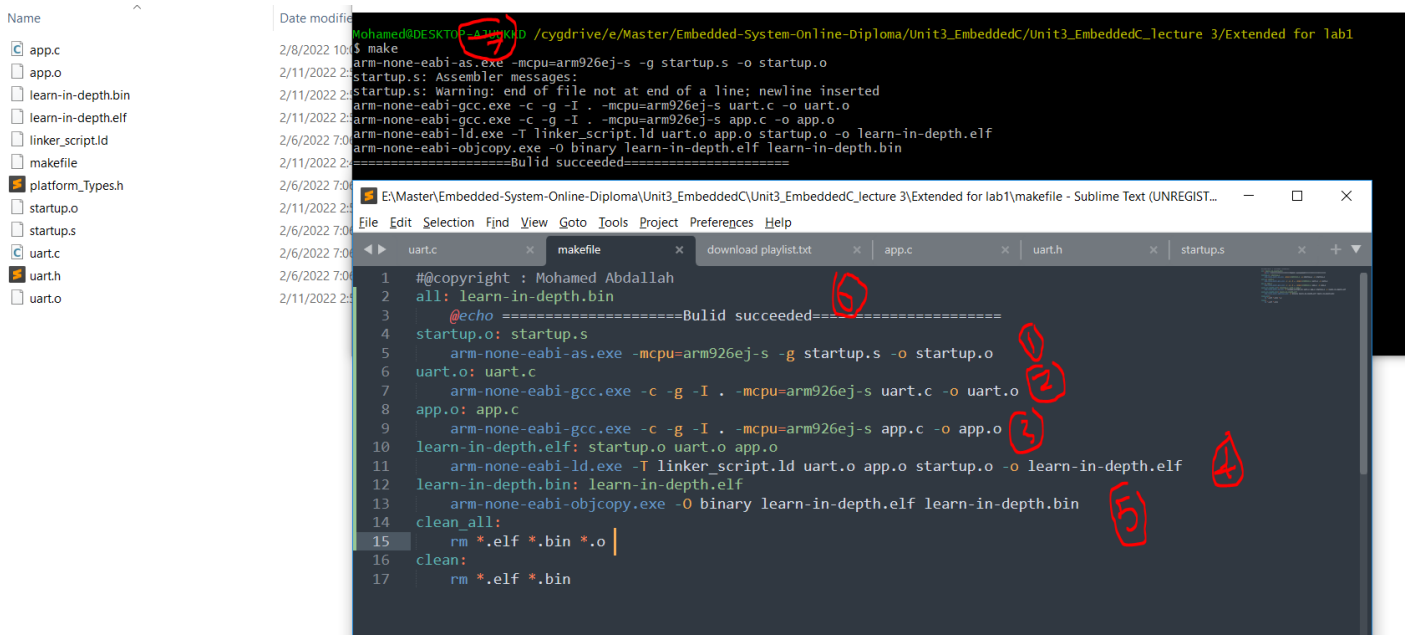


- Now he displayed current file and you can debug using GUI.

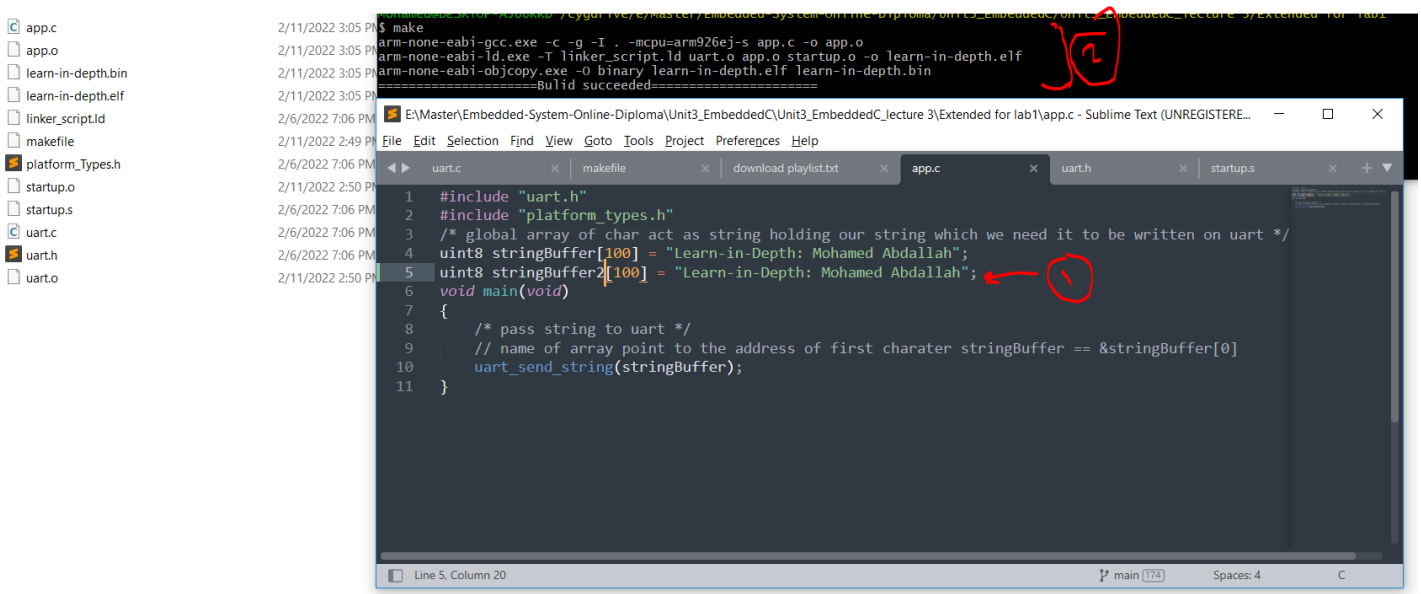
Makefile



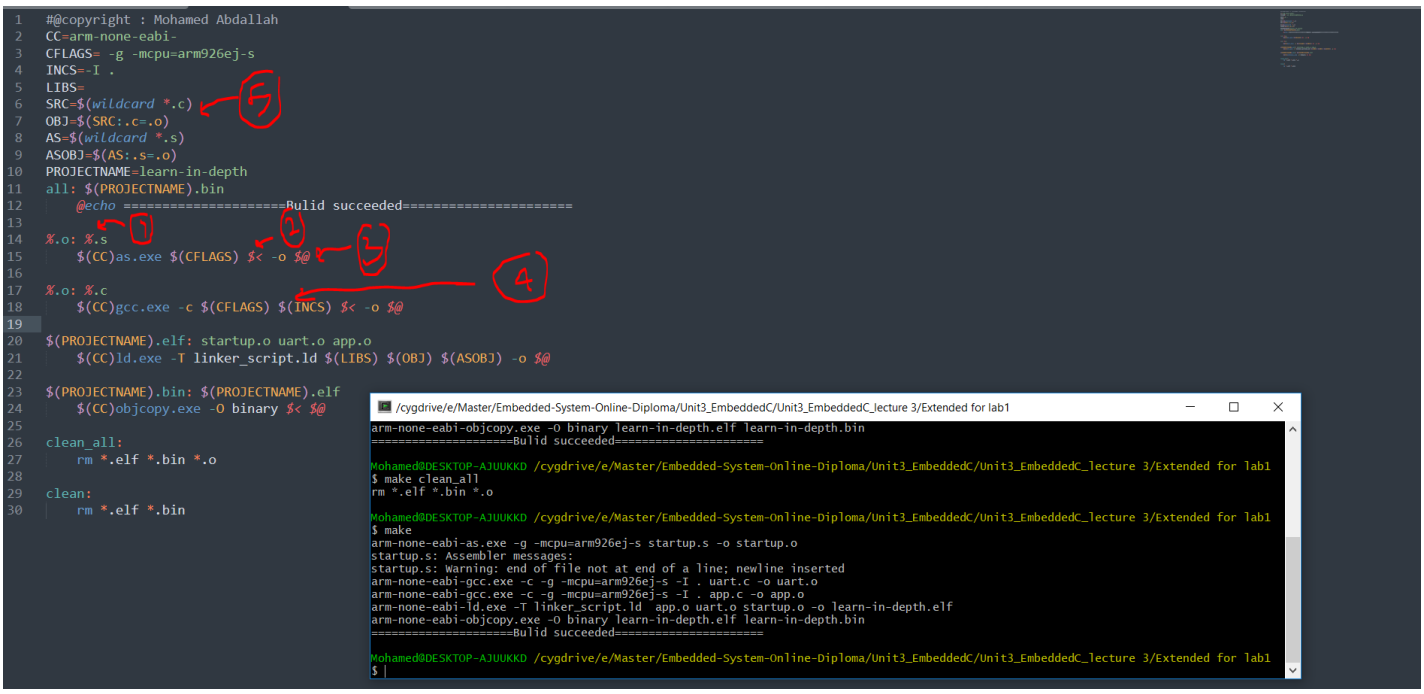
1. Clean_all → target, no dependences, role: remove all files .elf, .bin and .o.
2. How to call it? make then write target name, if we didn't write target it will execute first target.
3. startu.o → target, dependences → startup.s, role → assembler. Succeed



1. target → startup.o, dependences → startup.s, role → assembler.
2. Target → uart.o, dependences → uart.c, role → compile but not link (-c).
3. Same as 2.
4. Target → learn-in-depth.elf, dependences → startup.o, uart.o, app.o role → link.
5. Target → learn-in-depth.bin, dependences → learn-in-depth.elf, role → obtain img to burn on board (without debugging info).
6. As when we call make, the first target will be executed we put all target and made it depended on .bin so it will generate all missing dependences files.
7. Call make it generated all files and build succeeded as show.



1. We will make some changes on app.c file as shown.
2. Then build again we notice that it build app.c only then link again and generate binary image finally.
   (changed files only will be build and all files depended on that changed files as shown not all file).



```
#@copyright : Mohamed Abdallah
CC=arm-none-eabi-
CFLAGS= -g -mcpu=arm926ej-s
INCS=-I .
LIBS=
SRC=$(wildcard *.c)
OBJ=$(SRC:.c=.o)
AS=$(wildcard *.s)
ASOBJ=$(AS:.s=.o)
PROJECTNAME=learn-in-depth
all: $(PROJECTNAME).bin
    @echo ====================Bulid succeeded====================

%.o: %.s
    $(CC)as.exe $(CFLAGS) $< -o $@

%.o: %.c
    $(CC)gcc.exe -c $(CFLAGS) $(INCS) $< -o $@

$(PROJECTNAME).elf: startup.o uart.o app.o
    $(CC)ld.exe -T linker_script.ld $(LIBS) $(OBJ) $(ASOBJ) -o $@

$(PROJECTNAME).bin: $(PROJECTNAME).elf
    $(CC)objcopy.exe -O binary $< $@

clean_all:
    rm *.elf *.bin *.o

clean:
    rm *.elf *.bin
```

```
/cygdrive/e/Master/Embedded-System-Online-Diploma/Unit3_EmbeddedC/Unit3_EmbeddedC_lecture 3/Extended for lab1
arm-none-eabi-objcopy.exe -O binary learn-in-depth.elf learn-in-depth.bin
====================Bulid succeeded====================

Mohamed@DESKTOP-AJUUKKD /cygdrive/e/Master/Embedded-System-Online-Diploma/Unit3_EmbeddedC/Unit3_EmbeddedC_lecture 3/Extended for lab1
$ make clean_all
rm *.elf *.bin *.o

Mohamed@DESKTOP-AJUUKKD /cygdrive/e/Master/Embedded-System-Online-Diploma/Unit3_EmbeddedC/Unit3_EmbeddedC_lecture 3/Extended for lab1
$ make
arm-none-eabi-as.exe -g -mcpu=arm926ej-s startup.s -o startup.o
startup.s: Assembler messages:
startup.s: Warning: end of file not at end of a line; newline inserted
arm-none-eabi-gcc.exe -c -g -mcpu=arm926ej-s -I . uart.c -o uart.o
arm-none-eabi-gcc.exe -c -g -mcpu=arm926ej-s -I . app.c -o app.o
arm-none-eabi-ld.exe -T linker_script.ld  app.o uart.o startup.o -o learn-in-depth.elf
arm-none-eabi-objcopy.exe -O binary learn-in-depth.elf learn-in-depth.bin
====================Bulid succeeded====================

Mohamed@DESKTOP-AJUUKKD /cygdrive/e/Master/Embedded-System-Online-Diploma/Unit3_EmbeddedC/Unit3_EmbeddedC_lecture 3/Extended for lab1
$
```

1. %.s → anyfile.s.
2. $< → replaced with dependences file.
3. $@ → replaced with target file.
4. $(INC) -→ replaced with value of the variable.
5. Wildcard *.c → all files.c.
6. SRC:.c=.o → replace .c with .o in files of SRC.